8. Detailed plans

19 - sch413

Consultant:
Balázs Goldschmidt

Team members

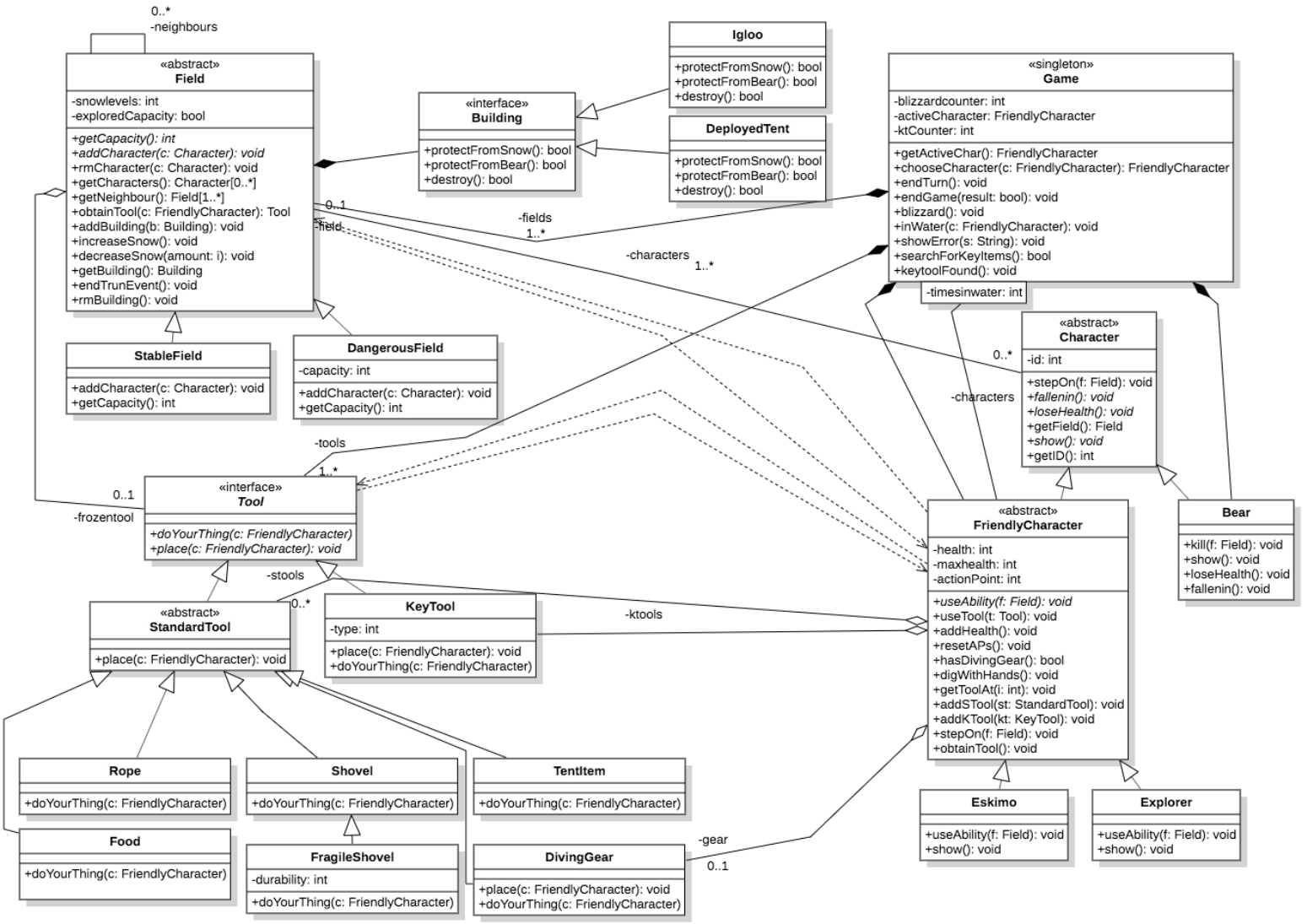
Bernát Ágó	VKMZ8A	ago.bernat216@gmail.com
Bankó Márton Bendegúz	AF1R7P	banko.marton@gmail.com
Balázs Mark Hain	YZMF1G	hainbalazs@gmail.com
Csaba Juhász	SDJI0N	csabszi.juhasz@gmail.com
Tamás Lukács	H4LUX2	lukacstomi1@gmail.com

2020.04.12.

8. Detailed plans

8.0 Changes compared to the previous submission

8.0.1 Class diagram



8.0.1 Descriptive language

The fields can be arranged in any order. All options must be mandatory for the commands, and the order of the options must be mandatory to match the order of the documentation. The new options listed here are added to the end of the option lists.

- rand command deleted
- body command added. Usage: type a number after the command name, then run the test with the specified number of lines. e.g. "test 2"
- create_game command: no size option, does not create fields. New option: blizzard: you can set the number of rounds until the storm comes. With random events turned off

there will always be a storm every so often. New option: rand this replaces the rand command, values can be 0,1

- keytoolsfound command deleted
- activecharacter command deleted
- set_field renamed to create_field, creates the given field. It should be issued immediately before create_char commands.

  New options:

  igloo/tent option replaced by: building, values: null, igloo, tent

  - tool_category: can be s, k, null, specifies the type of tool on it (standard, keytool, none)
  - tool_type: can be 1, 2, 3 for a key item, rope, food, shovel, fshovel, tent, dgear for other items.
  - durability: fshovel is the value that matters, it can be between 0 and 3

- addstdtool: location option removed, only adds a character, type option added, which tells the type of the object. Values: rope, food, shovel, fshovel, tent, dgear
- addktool: location_type deleted, can only give location_type to character
- create_char: You can also specify bear in the type option, in which case the value specified in the health option will be ignored.
- create_bear command deleted
- create_field: replace igloo/tent option with building, values: 0,

igloo, tent New commands to be displayed when interacting with the

character:

- obtain: The currently active character digs up the object frozen in his own field and takes it, if it is available.
- dig: The currently active character digs a layer of snow from his own field with his hand.
- skip: The currently active player passes his turn, the next player in line becomes active.

Changes made to the descriptive language of the kiement:

- **field_info** has been added to the description
  <building> attribute describing the type of building in the field (values: null/igloo/tent)
  <snow_level> is an attribute indicating the amount of snow in the field (values: int+)
  New descriptor signature: `<field_id><capacity_explored><capacity><standing_on [..]><neighbours [..]><frozen_tool><building><snow_level>`

- char_info description
  char_type is included: the given character type [eskimo/explorer/bear]
  the **in_water** attribute is omitted
  The new signature of the descriptor:
  `<char_id><field_id><health><actionpoints><tools [..]><char_type>`

- **dump_all** added in_water attribute as last global attribute, which lists characters that are in water in `<char [..]>` format New signature for the descriptor:

```
Number of fields
[m] Number of
players [s]
```

```
Next player number [char_id] Storm expected
in this many rounds: [t]
The game is still on / The game is over [c/w/l]
Checking out players in the water <char [..]>
for each character: char_info
for each field: field_info
```

8.1 Class and method designs.

8.1.1 Character

- **Responsibility**
  This abstract class represents the characters that can be placed on the fields in the game, and they can interact with the fields in different ways: they can move between two adjacent fields, fall off them into the water, or be victims of snowfall on the field. It knows and manages your status: what your ID is, which field you are standing on.
- **Ancestral classes: -**
- **Interfaces: -**
- **Attributes: id**
  [int]
  This is the unique identifier of the character, which can be used to reach him or her. Its value is constant and is assigned at the beginning of the game.
  gRef [Game]
  Direct knowledge of the Game singleton makes it much easier to communicate messages such as death or drowning.
  standingOn [Field]
  This attribute specifies which field the character is currently in.
- **Methods**

  void stepOn(f: Field)
  It moves the character to the field received as a parameter space, i.e. it changes the attribute of the field, and then it also signals to the field that it should accept a new character and removes it from the previous one.

  ```
  ha(f adjacent to standingOn)
        remove the character from standingOn, place
        the character on f,
        change the value of standingOn to f
  ```

  void fallIn()
  It will be called if for some reason the character would be in the water. Abstract function, some people can swim. If he gets in the water, he signals this to the Game.

  Field getField()
  The method returns with a reference to the field the character is standing on or has waterfalled from.

void loseHealth()
This method can be called to indicate the reaction of a given Character to the weather conditions. Abstract function, some characters tolerate the cold differently. Can be called after snowfall.

int getID()
Use this function to get the character ID.

void show()
Depending on the View, it displays information about the character. Abstract function. On prompt start, prints its properties in the format defined by *char_info.* Plots itself on gui.

8.1.2 FriendlyCharacter

- **Responsibility**
  This abstract class represents characters in the game that players can interact with. They know and manage their own state, e.g. how much life they have, how many actions they can still take, they store the tools they have acquired (Tool), they can use these and their own abilities. In addition, characters can use their own hands to remove a layer of snow from the field.
- **Ancestral classes: character**
- **Interfaces: -**
- **Attributes:**
  health [int]
  This attribute indicates how much life the character has at a given moment.
  actionPoint [int]
  This value determines how many more activities the character can perform in the given round. At the beginning of each turn, the character takes a value of 4, and then decreases this number by 1 for each activity.
  gear [DivingGear]
  Reference to the character's existing or non-existing (null) wetsuit.
  field [Field]
  This attribute specifies which field the character is currently in.
  ktools [KeyTool]
  The character stores its recorded key items in this attribute.
  stools [StandardTool]
  The character stores its recorded standard objects in this attribute.

- **Methods**
  void stepOn(f: Field)
  It moves the character to the field received as a parameter space, i.e. it changes the attribute of the field, and then it also signals to the field that it should accept another character, the

and removes it from the previous one.
Redefines the action described in Character by subtracting an action point.

```
ha(f adjacent to standingOn)
      remove the character from standingOn, place
      the character on f,
      change the value of standingOn to f

reduce the number of action points by 1
if(action points are exhausted)
      completes the player's turn
```

void fallIn()
If the field you would move to can no longer hold more players, this method is used to signal to the Game that the player is in water.

void useAbility(f: Field)
An abstract method that is responsible in descendants for ensuring that a character of a given type uses the appropriate ability. (In our case, the Eskimo builds igloos and the polar explorer measures capacity.) In the end, it uses an action point.

void useTool(t: Tool)
On the Tool interface object received as a parameter, it calls the method to use it and uses an action point.

void obtainTool()
The character digs out the frozen device from the field he is standing on, if something is frozen. If the digging was successful, it costs 1 work point, if it runs out of work points its turn ends.

```
try to get the frozen asset from the standingOn field if you have
acquired something
       reduce the number of action points by 1 if(action
      points are exhausted)
            completes the player's turn
```

Field getField()
The method returns the reference of the field the character is currently standing on.

void loseHealth()
This method can be called to indicate the reaction of a given Character to the weather conditions. Reduce the FriendlyCharacter's life by 1 when it runs out. Can be called after a snowfall.

void addHealth()
This method gives the player extra life in case the player does not exceed the maximum allowed.

void resetAPs()
After the player's turn, the number of action points available is expected to decrease slightly. To get it back to 4 in the next round, this method must be called.

bool hasDivingGear()
Returns true if the character has a wetsuit, false if not.

void digWithHands()
The character clears a layer of snow from the field where he is standing in exchange for a working point.

```
reduce the number of snow layers in the standingOn field
by 1 reduce the number of action points by 1
      if (action points are exhausted) the
            player's turn is completed
```

Tool getToolAt(i: int)
Returns the device stored in the character as i. For 0<=i<3 we return KeyTool, for i<=2 we return StandardTool

void addSTool(st: StandardTool)
Places a standard device in the character.

void addKTool(kt: KeyTool)
Places a key item in the character.

void removeFood(f: Food)
The food received as a parameter is removed from the character's own assets.

8.1.3 Eskimo

- **Responsibility**
  This class represents the Eskimos in the game, who are Friendly Characters. Its ability is to build igloos, which it can only do on its own field if it has snow on it at the time. It has a maximum life of 5.
- **Ancestor classes:** character ->FriendlyCharacter-> Eskimo
- **Interfaces: -**
- **Attributes: (inherited from Character: id, standingOn, gRef)**
- **Methods:**

  void useAbility(f: Field):
  This is the method he uses to build or destroy the igloo in the field he is standing on,

by calling the appropriate method on it. The igloo is not necessarily built, as it can only be done on a snowy field, and the destruction is always successful. It also deducts one action point at the end.

```
if (we don't want to use the ability on our own field)
      we throw an error message to the game about this
try to change the igloo state if(the change was
successful)
reduce the number of action points by 1
      if(action points are exhausted)
            completes the player's turn
by the way
      we throw an error message to the game that the igloo state
cannot be changed
```

void show()
Depending on the View, it displays information about the character. Abstract function. On prompt startup, prints its properties to the
<char_id><field_id><health><actionpoints><tools [..]><character_type>

format. In the case of Gui, it draws itself.

8.1.4 Explorer

- **Responsibility:**
  This class represents the polar explorers in the game, who are FriendlyCharacters. They have the ability to explore the ground, which they can only do on their own or adjacent fields. Maximum life is 4.
- **Ancestor classes:** character ->FriendlyCharacter-> Explorer
- **Interfaces: -**
- **Attributes: (inherited from Character: id, standingOn, gRef)**
- **Methods:**

  void useAbility(f: Field):
  This method determines the capacity of the field given as a parameter. It is placed in an action point. If a non-adjacent or own field is passed, it cannot determine it, but no action point is deducted.

```
if (we don't want to use the ability on our own or a neighbouring
      field) we throw an error message to the game about this
discover the capacity of the field
```

  void show()
  Depending on the View, it displays information about the character. Abstract function. On prompt startup, prints its properties to the
  <char_id><field_id><health><actionpoints><tools [..]><character_type>

  format. In the case of Gui, it draws itself.

8.1.5 Bear

- **Responsibility**
  This class represents the polar bears in the game, a character who moves one direction (randomly chosen by the Game) per turn. If you share an ice board with a friendly character that is not in a protected space, you will capture it and the game is over. Falling into water doesn't affect him, because he can swim out to another ice floe and catch other friendly characters in the water. Snowfall doesn't affect him either.
- **Ancestral classes: character**
- **Interfaces:**
- **Attributes: (inherited from Character: id, standingOn, gRef)**
- **Methods:**
  void fallIn()
  It will be called if for some reason the character would be in the water. The polar bear can swim, he is not counted. It is a direct empty method.
  void loseHealth()
  This is how the character reacts to the weather conditions. A polar bear used to snowfall does not lose body heat. It can be called after a snowfall. Directly empty method. **void kill()**
  A function is called for each move of the bear, which kills any unprotected (even water) players standing on it. If someone dies, the game ends. It is the Game's responsibility to call this.

  ```
  if(standingOn is not protected and there are more than
        1 players on the field) the game ends
  ```

  void show()
  Depending on the View, it displays information about the character. Abstract function. On prompt startup, prints its properties to the
  <char_id><field_id><character_type>

  format. In the case of Gui, it draws itself.

8.1.6 *Game*

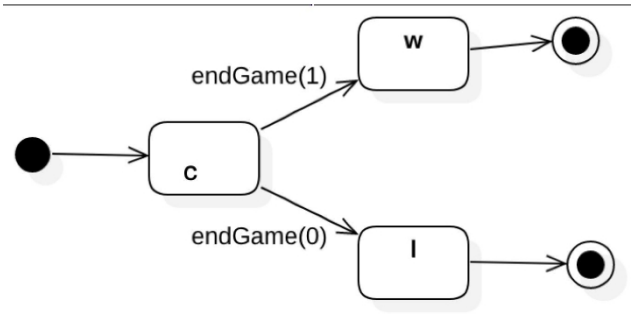
- **Responsibility:**

The class that implements the engine of the Model architecture. It stores the elements of the game. It handles incoming calls from the controller, keeps track of who is the next character to interact with. It manages the rounds, the turns and the events that occur at the end of them (snowfall, keeping track of players in water), checks if characters are dead. When rescuing with a rope, manages the choice of characters that can be rescued. It stores the number of key items found so far, so that when they are used it can decide if the player has won. Starts and ends the game.
Only one copy can exist.

- **Ancestral classes: -**
- **Interfaces -**
- **Attributes:**

ktCounter [int]:
It keeps an account of the key items that have been dug up.

gameState [char]:
Keeps track of the game status c: in progress, w= won, l= lost State machine:



characters [Character[1..*]]:
Storage of characters in the game.

fields [Field[1..*]]:
Storage of fields in the game.

bears [Bear[1..*]]:
Store the bears in the game.

activeCharacter [Character]:
Reference to the next character in the sequence (from the current characters), you can interact with it.

blizzardCounter [int]:
After a snowfall (within a given range), a randomly generated counter indicates how many rounds until snowfall is expected again.

timesinwater [Character, int]:
A hashmap keeps track of the characters that are currently in the water, and how long they can stay in the water.

deafultBlizzard [int]:
If it is 0, blizzardCounter will get random values, if it is positive, it will always get the same value after a storm.

rand [Random]:
For random events in the game.

randstate[boolean]:
Stores whether random events are allowed.

Field getField(int):
Returns the field with the given id, if there is none it returns ShowError to indicate that an error occurred.

Character getCharacter(int):
Returns the character with the given id, if there is no id it returns ShowError to indicate that an error occurred.

- **Methods:**

Game(int dbc, boolean rand):
Initializes random variable, blizzardCounter, defaultBlizzard and randstate.

Character getActiveChar():

Returns a reference to the character that is currently in line, so it just returns activeCharacter.

void endTurn():
Handles end-of-round events: sets new active character, checks if characters that have been waterboarded die, calls endGame if they do. If the blizzardCounter expires, i.e. the storm is due, it calls blizzard(). It also interacts with the bears on the field.

pseudocode:

```
activeCharacter.resetAPs
Blizardcounter = blizarcounter-1
If blizzarcounter = 0 then
```

call the blizzard function and
if randstate = false then blizzardCounter = defaultBlizzard else
blizzardCounter = random(0-> characters size)

All m bears from the bears list:
Field f
If randstate = false then f = m.getNeighbour(0)
otherwise f = random neighbour
m.stepOn(f)
m.kill

For each f field from the fields list:
f.endTurnEvent

For each c from the characters list:
If there is a value for c in the timesInWater hashmap then the value is decreased by one.
If the value is 0 and the hasDivingGear function called on c returns false, then the EndGame function is called with false

activeCharacter = next Character in the list of characters, or back to the beginning if there are no more

void blizzard():

This function is called during a storm. It randomly selects fields from the ice field, increases the snow level of these fields, and subtracts one life from characters not in a building.

pseudocode:

For each f field from the fields list:
If randstate = false or random(2) = 0
f.increaseSnow
If f.getBuilding.ProtectFromSnow = false then for each character c from the f.getCharacters list
c.loseHealth

void endGame(bool result):

The end of game function, tells the user that the game is over and with what result. Its implementation is controller dependent. In a console environment, it just tells the user whether he won or lost. It changes the value of the gameState attribute according to the input.

void showError(String s):
Some interactions are only allowed under certain circumstances, if these are not present, the activity cannot be performed and the user will be notified via the Game. In a console environment, it just prints the string received.

void inWater(Character c):
It is informed that character c is watered, so it updates timesInWater, i.e. it adds the character and assigns the number of characters as value.

Character chooseCharacter(Character c):
This call is the result of the work done by Rope. In this case, the Game will cross out the people who have been waterlogged in the character field (circle) and return with the one of their choice (chosen by the user).

pseudocode:

```
bool succes = false
amíg(succes = false)
  print("Which character did you save?")
  List(Character) cl
  For each field f from c.getField().getNeighbour() For
          each character c2 from f.getCharacters
                  If there is a value for c2 in timesinwater,
                          print(c2.getId)
  I = scanned number
  For each c3 character from the characters
          list if I = c3.getId()
          succes = true
          return c3
  ShowError("Failed to select character")
```

void keyToolfound():
Increases the number of keys found by one.

void show():
It prints the values of the game attributes, calls the list elements fields, characters and bears, and calls the functions that print the list elements.

pseudocode:

```
Print(fields size)
Print(characters size)
If activeChar is not null Print(activeChar.getId)
Print(blizzardCounter)
Print(gameState)
For each character c from the characters list:
c.show For each bear b from the bears list:
b.show For each field f from the fields list: f.show
```

Boolean searchForKeyItems():
It checks that all the requirements to win the game are met. This consists of checking that all characters are on the same field and that all key items have been found. If so, the game is over. If we have won the game true, otherwise false is the return value.

pseudocode:

```
Field f = characters(0).getField()
If ktCounter != 3 return false
For each character c in the characters list
  if c.getField() != f return false;
Return true
```

Setters needed to set the game state:

void addChar(Character):
Adds a character to the characters list. If this is the first character, it sets activeCharacter to

void drowning(int id, int timeleft)
Select the character with the given identifier from the list of characters and add the corresponding value to timesInWater

void addBear(Bear)
We add the bear to the bears list.

void addNeighbour(int, int)
Sets the adjacency between two fields.

void addField(Field)
Adds the field to the fields list

The following classes are not an integral part of the model, but they have an important additional role in the skeleton and prototype:

8.1.7 SchLogger

- **Responsibility**

  This department is responsible for project logging. It is able to record the exit and entry points of methods, and also handles questions before user inputs

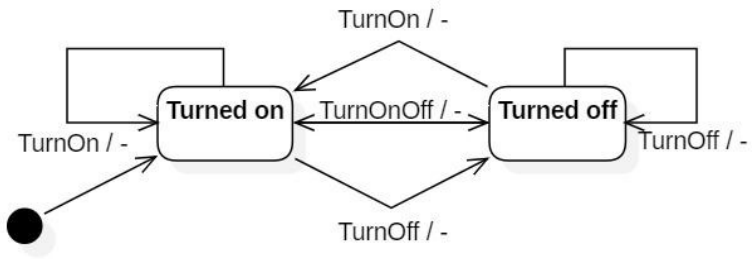
- **Attributes**

-ch [ConsoleHandler]: this object comes from the java util logging library, we use it to connect our logger to our formatter.
-on [boolean]: a plain boolean variable: if true, the logger works, if false, it does nothing on calls.

- **Methods**

+void setup():
This method initializes the logger created in main. We tell it to accept log messages of all priorities, and we assign CsFormatter, which then handles logging at the method's entry and exit points.
+String opt(String[] str):
This method is responsible for writing out questions before user input in the skeleton. We first set up an OptFormatter for this purpose, and then pass the string array to it. Afterwards, to avoid the user having to deal with this, we reset the default CsFormatter as our Logger's Formatter + Requests input from the user after the questions are printed
+void turnOn():
It "turns on" our logger, i.e. from this point onwards it accepts all priority messages. On is set to true.
+void turnOff():
"Turn off" our logger, i.e. from this point onwards it will not accept any priority messages. On is set to false.
+void turnOnOff():
It "toggles" our logger, i.e. if it was off, it turns it on, otherwise it turns it off.

- **State chart diagram:**



8.1.8 CsFormatter

- **Responsibility**
  This Formatter is responsible for writing log messages to and from methods in the correct format.
- **Classes**
  java.util.logging.Formatter -> CsFormatter
- **Attributes**
  -stack_depth[Integer]:
  The stack_depth attribute keeps track of how deep we are in the method call (this is needed for indentation).
- **Methods**
  +String format(Logrecord rec):
  This is an override.
  The >> or << passed in the logger.info() method helps us to decide whether we have now moved down or up the stack. We increase or decrease the stack_depth accordingly. Then, knowing the depth, the formatting can be done, the rec parameter contains all the relevant information.

8.1.9 OptFormatter

- **Responsibility**
  This Formatter is responsible for writing out user input requests in the skeleton in the correct format.
- **Classes**
  java.util.logging.Formatter -> OptFormatter
- **Attributes**
  -lastmessage[String[]]:
  This attribute stores the questions and answers to be printed.
- **Methods**
  +String format(Logrecord rec):
  This is an override.
  Lastmessage is used to prepare the message to be printed according to the specified formatting.
  +void opt(String[] str):
  Here we specify the value of the lastmessage, which the format will be able to easily rewrite.

8.1.10 SchParser

- **Responsibility**
  This class is responsible for correctly processing incoming messages in the prototype and for starting the appropriate processes.
- **Attributes**
  -g[Game]:

This object represents the Game object created in the main, through which all important events happen.
-pw[PrintWriter]:
This object is responsible for handling the output file.
-active[Integer]:
In test cases where there is a file to compare the output with, we can specify its serial number here.

- **Methods**

  For each command, there is a method with the same name, which handles the corresponding tasks. The others are:
  +void getInput():
  It waits for instructions from the console, and when a line arrives, it passes it straight to the processing method.
  -void forwarder(String line):
  This method splits the incoming command into units along the spaces, and then passes it to the appropriate method based on the instruction type.
  +compare()
  If the output has been written to a file, it compares the correctness of the output with a pre-compiled "control" file, if such a file exists.

8.1.11 Field

- **Responsibility**

  This abstract class represents the ice sheets that make up the ice field. It knows and manages its own state, e.g. whether it has snow on it, whether it has buried tools on it, whether it has igloos built on it, etc. Fields know the adjacent Fields, they can be queried.

- **Attributes:**

  building [Building]
  Reference to the building in the field, if not on it, then zero.

  snowlevels [int]
  The snowlevels attribute determines how many layers of snow are currently on the field, this value will never be greater than 5. The value can change with snow shovelling, igloo building or the arrival of a storm.

  exploredCapacity [bool]
  During the game, a polar explorer can determine the capacity of the field (whether it is stable, if not, how many people it can hold). This attribute indicates whether it has already been done. Once it becomes true, it remains true until the end of the game.

  neighbours [Field[0..*]]
  This collection contains the fields adjacent to a field, and depending on the shape and location of the field, it can have as many neighbours as you like.

is not worth more than 6). This attribute is constant throughout the game and contains the same fields.

characters [Character[0..*]]
This collection contains the characters currently in the field. A new character is added when someone steps on it, or when someone is pulled onto the field from the water, a character is removed when it steps off or is pulled onto another field.

frozenTool [Tool]
This attribute specifies the buried device (if there is a buried device in the given field)

- **Methods**

  int getCapacity():
  Abstract method that returns the capacity of a given field. A stable field and a dangerous field return different results, hence the need for abstraction.

  void addCharacter(c: Character):
  An abstract method that places a character in the field, given as a parameter. Abstraction is necessary because in the case of a dangerous field, you need to check if the capacity is exceeded, whereas in the case of a stable field, everything is fine.

  void rmCharacter(c: Character):
  If a character escapes from a field, it is indicated to the field by this function. The avoiding character is passed as a parameter.

  Character[] getCharacters():
  This method is used to retrieve the collection of characters currently in the given field.

  Field[] getNeighbour():
  This method is used to query the collection of fields adjacent to a given field.

  Tool obtainTool():
  This method resolves the excavation of a device frozen in a field, and returns a message if the field does not contain a device. If an EquippableTool is frozen in the field, it will be dug out and placed in the character's toolbar.

  void addBuilding(b: Building):
  If the value of the building attribute is null, the specified building will be added to the building attribute. Otherwise it does nothing.

  void rmBuilding():
  Sets the building attribute of the field to 0.

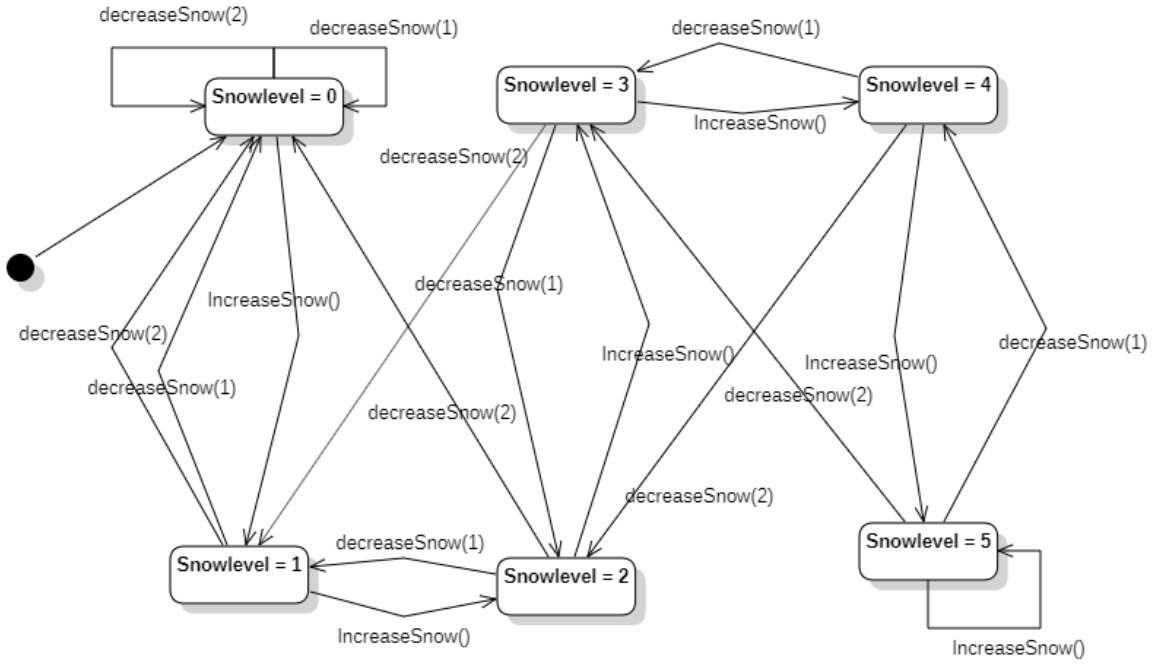
void increaseSnow():
This method increments the snow layer in the field by 1 if it has not yet reached 5, and leaves it at 5 if it has.

void decreaseSnow(amount: int):
This method reduces the snow layer in the field by the value specified in the parameter if there is still a snow layer in the field.

Building getBuilding():
Returns the building in the field.

void endTurnEvent():
It makes the changes to the field that the game specification says should happen at the end of the round.

- **State chart diagram:**



8.1.12 StableField

- **Responsibility**

  This class represents the stable fields that make up the ice field, i.e. fields from which it is not possible to fall into the water.

- **Classes**

  Field

- **Attributes**

  capacity [int]
  This constant attribute specifies how many characters the field can hold.

- **Methods**

  int getCapacity():
  This method returns the capacity of the given field.

  void addCharacter(c: Character):
  This method places the character that is passed to the field as a parameter. It checks if the capacity has been exceeded, if so, it signals to the Game class that the characters on it have been flooded.

8.1.13 DangerousField

- **Responsibility**

  This class represents the dangerous fields that make up the ice field, i.e. fields from which it is possible to fall into the water

- **Classes**

  Field

- **Attributes**
  - **-**

- **Methods**

  int getCapacity():
  This method returns -1, indicating that it has no capacity interpreted.

  void addCharacter(c: Character):
  This method places the character received as a parameter in the field, i.e. it puts it into the characters array.

8.1.14 Building

- **Responsibility**

Abstract class. This symbolizes a building on the field, it is his responsibility to let the other classes know how his presence there affects the game.

- **Attributes**
  - -
- **Methods**
  - **bool protectFromSnow()**: Returns whether the building protects the characters in the field from snowstorms.
  - **bool protectFromBear()**: Returns whether the building protects the characters on the field from bear attack.
  - **bool destroy():** Returns whether the building should be destroyed at the end of the round.

8.1.15 Igloo

- **Responsibility**

  It is an igloo, a structure that protects characters from both blizzards and bears.

- **Interfaces**

  Building

- **Attributes**
  - -
- **Methods**
  - **bool protectFromSnow()**: returns true, because igloo always protects from snowstorms.
  - **bool protectFromBear()**: returns true, since igloo always protects from bear attack.
  - **bool destroy():** returns false, because the igloo does not disappear at the end of the round.

8.1.16 DeployedTent

- **Responsibility**

  This is a tent, a structure that protects characters from blizzards, but disappears after one round.

- **Interfaces**

  Building

- **Attributes**
  - -
- **Methods**
  - **bool protectFromSnow()**: returns true, because the tent always protects from snowstorms.
  - **bool protectFromBear()**: Returns false, since the tent never protects against bear attack.

- **bool destroy():** returns true, since the tent disappears at the end of the round.

8.1.17 DivingGear

- **Responsibility:**
  It is a standard device that has a passive capability, if the character has it, it can survive in the event of a waterfall without being pulled out.
- **Ancestor classes:** tool -> StandardTool
- **Interfaces: -**
- **Attributes: -**
- **Methods:**

  Diving Gear(Game game game):
  Basic constructor to set the reference to the Game.
  void doYourThing(FriendlyCharacter c):
  It does nothing, it has no active capacity.
  void place(FriendlyCharacter c):
  Places itself in the character's divingGear attr. via the setDivingGear function.
  String show():
  Function used for testing, debugging. Returns the name of the object (DivingGear).

8.1.18 KeyTool

- **Responsibility:**
  A key item to get. There are only 3 items (FlareGun, Cartridge, Flare) in the whole game. Their ability is to assemble and fire, so they start searching for key items on the same field, if all 3 are found on 1 field, they fire the flare gun, the game ends in victory. They are stored in a separate place in the toolbox.
- **Ancestry classes:** tool
- **Interfaces: -**
- **Attributes:**
  type [int]
  Tell which of the 3 objects you can find is the one you are talking about

- **Methods:**

  KeyTool(int type, Game game):
  A basic constructor to set the reference to the Game and the type of the key object (1-3).
  void doYourThing(FriendlyCharacter c):
  Calls the Game searchForKeyItems() function, which checks the conditions needed to win the game
  void place(FriendlyCharacter c):

It is inserted in the toolbar of the character received as a parameter, in the space reserved for key objects. After that, it tells the Game that a key item has been obtained.

String show():

Function used for testing, debugging. Returns the name of the object (KeyTool) and the type of the keyTool, separated by commas.

8.1.19 Food

- **Responsibility:**
  A standard item that the character can consume, regenerating 1 body heat (but not exceeding the maximum). Once the food is consumed, the item disappears from the toolbar.
- **Ancestor classes:** tool -> StandardTool
- **Interfaces: -**
- **Attributes: -**
- **Methods:**

  Food(Game game):

  Basic constructor to set the reference to the Game.

  void doYourThing(FriendlyCharacter c):

  Increases the character's body heat by 1 (but cannot exceed the maximum body heat). Then calls the character's removeFood(Food) function, passing itself as a parameter

  String show():

  Function used for testing, debugging. Returns the object name (Food).

8.1.20 Rope

- **Responsibility:**
  A standard tool that allows a character to pull a waterlogged companion from an adjacent field.
- **Ancestor classes:** tool -> StandardTool
- **Interfaces: -**
- **Attributes: -**
- **Methods:**

  Rope(Game game):

  Basic constructor to set the reference to the Game.

  void doYourThing(FriendlyCharacter c):

  Select a character in an adjacent field if possible. It does this by calling the chooseCharacter() method on the Game, which if it does not return 'null', will drag the character to be selected to its own field.

  String show():

  Function used for testing, debugging. Returns the name of the object (Rope).

8.1.21 Shovel

- **Responsibility:**
  A standard tool that allows a character to remove 2 layers of snow from the field they are on at the same time, for one working point.
- **Ancestor classes:** tool -> StandardTool
- **Interfaces: -**
- **Attributes: -**
- **Methods:**

  Shovel(Game game game):
  Basic constructor to set the reference to the Game.
  void doYourThing(FriendlyCharacter c):
  Remove 2 layers of snow from the field where the player is standing (at most if possible).
  String show():
  Function used for testing, debugging. Returns the name of the object (Shovel).

8.1.22 StandardTool

- **Responsibility:**
  An abstract class representing an obtainable tool that does not form an integral part of the game, i.e. it is not a key object.
- **Classes:** tool
- **Interfaces: -**
- **Attributes: -**
- **Methods:**

  StandardTool(Game game):
  Basic constructor to set the reference to the Game.
  void place(FriendlyCharacter c):
  Places the tool itself in the collection of the character tool it receives as a parameter, among the other standard tools.

8.1.23 Tool

- **Responsibility:**
  An abstract ancestor class representing tools. Each tool has a specific ability that the character (who owns it) can use. They cost 1 job point to use. It also has a reference to an object of the Game class.
- **Ancestral classes: -**
- **Interfaces: -**
- **Attributes:**

  gRef [Game]

Reference to the object of the Game class.

- **Methods:**

  Tool(Game game):
  Basic constructor to set the reference to the Game.
  void doYourThing(FriendlyCharacter c):
  (Abstract method.) The tool will do its job if the conditions are right.
  void place(FriendlyCharacter c):
  (Abstract method) Places the device itself in the collection of the character device given as a parameter, in the appropriate place.

8.1.24 FragileShovel

- **Responsibility:**
  A standard tool that is similar in function to a shovel, but breaks after three uses and you can only remove one layer of snow.
- **Ancestor classes:** tool -> StandardTool ->Shovel
- **Interfaces: -**
- **Attributes:**

  durability [int]
  Attribute indicating the use of the shovel.

- **Methods:**

  FragileShovel(Game game):
  A basic constructor to set the reference to the Game and set durability to
  3.
  void doYourThing(FriendlyCharacter c):
  Removes 2 layers of snow from the field where the player is standing (at most if possible), and its usability decreases by one. If durability reaches 0, only one layer of snow is removed.
  String show():
  Function used for testing, debugging. Returns the name of the object (FragileShovel) and its usage, separated by commas.

8.1.25 TentItem

- **Responsibility:**
  A standard tool that the owning character can use to build a tent on his own field.
- **Ancestor classes:** tool -> StandardTool
- **Interfaces: -**
- **Attributes: -**
- **Methods:**

TentItem(Game game):
Basic constructor to set the reference to the Game.
void doYourThing(FriendlyCharacter c):
If the tool is used, it tries to set up a new tent on the character field received in the parameter.
String show():
Function used for testing, debugging. Returns the name of the object (TentItem).

8.2 Detailed test designs, description in the language of the test

8.2.1 Eskimo Dies in Blizzard

- **Description**
  The snowstorm that arrives at the end of the round takes the last of the Eskimo's life points, killing him.
- **Checked functionality, expected failure points**
  What this test case verifies is that the character really only dies if there is no igloo or tent in the field, and that he had one life to begin with.
- **Bemenet**
  create_game -blizzard=1 -rand=0
  create_field -id=0 -capacity=-1 -snow=0 -explored=1 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_char -id=0 -type=eskimo -health=1 -field=0
  char_info -id=0
  skip
  dump_all
- **Expected output**
  <0><0><false><1><4><>
  1
  1
  0
  1
  l
  <>
  <0><0><0><0><><eskimo>
  <0><true><-1><0><null><null><1>

8.2.2 Explorer Drowns

- **Description**
  If the player is not released from the water for a certain period of time, he will die.
- **Checked functionality, expected failure points**
  This test case checks if the character actually falls into the water and only dies when in the water, and the time that the character can be in the water without a wetsuit has elapsed.
- **Bemenet**
  create_game -blizzard=5 -rand=0
  create_field -id=0 -capacity=0 -snow=1 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0

create_char -id=0 -type=explorer -health=4 -field=0
drowning -id=0
char_info -id=0
skip
dump_all

- **Expected output**
  <0><0><true><4><4><>
  1
  1
  0
  4
  l
  <0>
  <0><0><0><0><><explorer>
  <0><false><0><0><null><null><1>

8.2.3 Eskimo Runs Out Of APs, Ends Turn

- **Description**
  During the different activities (such as stepping, using tools, using skills), the energy points of the actor are constantly consumed. After a while, they reach 0, at which point the player's turn is over.
- **Checked functionality, expected failure points**
  In this test, the Eskimo spends all its action points, after each action we check if the action point has been successfully deducted, and at the end we check if it has been used up. We will monitor the success of its actions as we go along.
- **Bento**
  create_game -blizzard=5 -rand=0
  create_field -id=0 -capacity=-1 -snow=1 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_field -id=1 -capacity=-1 -snow=1 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
  add_neighbour -id1=0 -id2=1
  create_char -id=0 -type=eskimo -health=5 -field=0
  char_info -id=0
  ability -field=0
  char_info -id=0
  ability -field=0
  char_info -id=0
  move -to=1
  char_info -id=0
  ability -field=1
  dump_all
- **Expected output**
  <0><0><false><5><4><>
  <0><0><false><5><3><>
  <0><0><false><5><2><>
  <0><0><false><5><1><>
  2

1
0
4
c
<>
<0><0><5><0><><><eskimo>
<0><false><0><null><1><null><null><0>
<1><false><0><0><0><null><igloo><0>

8.2.4 Eskimo Skips Turn

- **Description**
  The player passes his turn to the next player before using up all his energy points.
- **Checked functionality, expected failure points**
  This test verifies that the circle transfer is working properly, i.e. that all the character's action points are deducted, the blizzard counter is reduced, etc.
- **Bemenet**
  create_game -blizzard=5 -rand=0
  create_field -id=0 -capacity=1 -snow=1 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_char -id=0 -type=eskimo -health=5 -field=0
  char_info -id=0
  skip
  dump_all
- **Expected output**
  <0><0><false><5><4><>
  1
  1
  0
  4
  c
  <>
  <0><0><5><0><><><eskimo>
  <0><false><0><0><null><null><1>

8.2.5 Digging without shovel

- **Description**
  The actor digs 1 layer of snow with his bare hands.
- **Checked functionality, expected failure points**
  Here we test whether, when a character digs in a field without a shovel, his action point is correctly deducted, whether the amount of snow is correctly reduced.
- **Bemenet**
  create_game -blizzard=5 -rand=0
  create_field -id=0 -capacity=1 -snow=2 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_char -id=0 -type=eskimo -health=5 -field=0

char_info -id=0
field_info -id=0
dig
dump_all

- **Expected output**
  <0><0><false><5><4><>
  <0><false><0><0><null><null><2> 1
  1
  0
  5
  c
  <0><0><5><3><><eskimo>
  <0><false><0><0><null><null><1>

8.2.6 Digging with shovel

- **Description**
  The actor uses a shovel to remove two layers of snow from his field
- **Checked functionality, expected failure points**
  Will two layers of snow really disappear from the field, or will there be 0 layers of snow after 1 layer. Does the number of energy points of the operator decrease.
- **Bento**
  create_game -blizzard=5 -rand=0
  create_field -id=0 -capacity=-1 -snow=3 -explored=1 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_char -id=0 -type=eskimo -health=3 -field=0
  addstdtool -location_id=0 -durability=0 -type=shovel
  char_info -id=0
  field_info id=0
  tool -id=0
  char_info -id=0
  field_info id=0
  tool -id=0
  char_info -id=0
  field_info id=0

- **Expected output**
  <0><0><3><4><{0, shovel}><eskimo>
  <0><true><-1><0><null><null><3>
  <0><0><3><3><{0, shovel}><eskimo>
  <0><true><-1><0><null><null><1>
  <0><0><3><2><{0, shovel}><eskimo>
  <0><true><-1><0><null><null><0>

8.2.7 Character eats Food

- **Description**
  The character eats a possessed food, which increases his life by one.

- **Checked functionality, expected failure points**
  Whether the character's life points increase if they are no longer at maximum. If the character's energy points decrease.
- **Bemenet**
  create_game -blizzard=5 -rand=0
  create_field -id=0 -capacity=-1 -snow=0 -explored=1 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_char -id=0 -type=eskimo -health=4 -field=0
  addstdtool -location_id=0 -durability=0 -type=food
  char_info -id=0
  tool -id=0
  char_info -id=0
  addstdtool -location_id=0 -durability=0 -type=food
  tool -id=0
  char_info -id=0

- **Expected output**
  <0><0><4><4><{0, food}><eskimo>
  <0><0><5><3><><eskimo>
  <0><0><5><2><><eskimo>

8.2.8 Stepping on full Dangerous Field

- **Description**
  The character enters an unstable field that is bound to collapse with the arrival of the new person.
- **Checked functionality, expected failure points**
  Whether the actor is transferred to the new field (whether the field knows that the actor is on it, whether the actor knows that it is on the field). Whether it is administered that actors on the encroaching field have fallen into the water.
- **Bemenet**
  create_game -blizzard=5 -rand=0
  create_field -id=0 -capacity=-1 -snow=3 -explored=1 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_field -id=1 -capacity=1 -snow=3 -explored=1 -building=null -tool_category=null -tool_type=shovel -durability=0
  add_neighbour -id1=0 -id2=1
  create_char -id=0 -type=eskimo -health=3 -field=0
  create_char -id=1 -type=eskimo -health=3 -field=1
  char_info -id=0
  char_info -id=1
  field_info -id=0
  field_info -id=1
  move -to=1
  dump_all

- **Expected output**
  <0><0><3><4><><eskimo>
  <1><1><3><4><><eskimo>

<0><true><-1><0><1><null><null><3>
<1><true><1><1><0><null><null><3>
2
2
0
5
c
<0,1>
<0><1><3><3><><eskimo>
<1><1><3><4><><eskimo>
<0><true><-1><1><null><null><3>
<1><true><1><0,1><0><null><null><3>

8.2.9 Stepping on a not full Dangerous Field

- **Description**
  The protagonist enters an unstable field that does not change with the arrival of a new person.
- **Checked functionality, expected failure points**
  Whether the actor is transferred to the new field (whether the field knows the actor is on it, whether the actor knows it is on the field). Whether the actor has lost energy points.
- **Bemenet**
  create_game -blizzard=5 -rand=0
  create_field -id=0 -capacity=-1 -snow=3 -explored=1 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_field -id=1 -capacity=2 -snow=3 -explored=1 -building=null -tool_category=null -tool_type=shovel -durability=0
  add_neighbour -id1=0 -id2=1
  create_char -id=0 -type=eskimo -health=3 -field=0
  char_info -id=0
  field_info -id=0
  field_info -id=1
  move -to=1
  char_info -id=0
  field_info -id=0
  field_info -id=1

- **Expected output**
  <0><0><3><4><><eskimo>
  <0><true><-1><0><1><null><null><3>
  <1><true><2><0><null><null><3>
  <0><1><3><3><><eskimo>
  <0><true><-1><1><null><null><3>
  <1><true><2><0><0><null><null><3>

8.2.10 Stepping on Stable Field

- **Description**

The player steps onto a stable field.

- **Checked functionality, expected failure points**
  Whether the actor is transferred to the new field (whether the field knows the actor is on it, whether the actor knows it is on the field). Whether the actor has lost energy points.
- **Bemenet**
  create_game -blizzard=5 -rand=0
  create_field -id=0 -capacity=-1 -snow=3 -explored=1 -building=null
  -tool_category=null -tool_type=shovel -durability=0
  create_field -id=1 -capacity=-1 -snow=3 -explored=1 -building=null
  -tool_category=null -tool_type=shovel -durability=0
  add_neighbour -id1=0 -id2=1
  create_char -id=0 -type=eskimo -health=3 -field=0
  char_info -id=0
  field_info -id=0
  field_info -id=1
  move -to=1
  char_info -id=0
  field_info -id=0
  field_info -id=1

- **Expected output**
  <0><0><3><4><><eskimo>
  <0><true><-1><0><1><null><null><3>
  <1><true><-1><0><null><null><3>
  <0><1><3><3><><eskimo>
  <0><true><-1><1><null><null><3>
  <1><true><-1><0><0><null><null><3>

8.2.11 Using Key Tool Correctly

- **Description**
  The player uses the key piece he has in the right circumstances, and the game is over, the players win.
- **Checked functionality, expected failure points**
  All the players (3 of them) are on a single field and all the key pieces are obtained (in this case, the holders are different). Then one of them uses his key piece, thus winning the game. Check to see if the game is completed and this is written out.
- **Bento**
  create_game -blizzard=5 -rand=0
  create_field -id=0 -capacity=-1 -snow=0 -explored=1 -building=null
  -tool_category=null -tool_type=shovel -durability=0
  create_char -id=0 -type=eskimo -health=3 -field=0
  create_char -id=1 -type=eskimo -health=3 -field=0
  create_char -id=2 -type=eskimo -health=3 -field=0
  addktool -location_id=0 -type=1
  addktool -location_id=1 -type=2
  addktool -location_id=2 -type=3

char_info -id=0
char_info -id=1
char_info -id=2
tool -id=0
dump_all

- **Expected output**
  <0><0><3><4><{1,ktool}><eskimo>
  <1><0><3><4><{2,ktool}><eskimo>
  <2><0><3><4><{3,ktool}><eskimo>
  1
  3
  0
  5
  w
  <>
  <0><0><3><3><{1,ktool}><eskimo>
  <1><0><3><4><{2,ktool}><eskimo>
  <2><0><3><4><{3,ktool}><eskimo>
  <0><true><-1><0,1,2><null><null><0>

8.2.12 Using Key Tool Incorrectly

- **Description**
  The player uses the key he has in the wrong circumstances (not all three keys are dug up), so the game is not over yet.
- **Checked functionality, expected failure points**
  Conditions for assembling/launching a rocket:
  - all three key items have been excavated,
  - all players are standing on the same field at the moment of launch,
  - no one is in the water
- **Bento**
  create_game -blizzard=1 -rand=0
  create_field -id=0 -capacity=-1 -snow=0 -explored=1 -building=false -tool_category=null -tool_type=0 -durability=0
  create_field -id=1 -capacity=-1 -snow=0 -explored=1 -building=false -tool_category=k -tool_type=3 -durability=-1
  create_char -id=0 -type=eskimo -health=1 -field=0
  addktool -location_id=0 -type=1
  create_char -id=1 -type=eskimo -health=1 -field=0
  addktool -location_id=1 -type=2
  create_char -id=2 -type=eskimo -health=1 -field=0
  tool -id=0

- **Expected output**
  The character couldn't assemble the rocket, it doesn't have all the elements!

8.2.13 Eskimo Obtains Shovel

- **Description**
  The Eskimo picks up a shovel from the field he is standing on.

- **Checked functionality, expected failure points**
  The success rate of digging out a given tool from a given field:
  the Eskimo is actually standing in a field where:
  - number of snow layers 0
  - no building
  (- the desired object is indeed frozen in the field)
  If these are not met, the message "The device cannot be excavated!" is displayed.
  If the excavation is successful, the tool is added to the character's imaginary toolbox with the highest number.
- **Bento**
  create_game -blizzard=1 -rand=0
  create_field -id=0 -capacity=-1 -snow=0 -explored=1 -building=false -tool_category=s -tool_type=shovel -durability=-1
  create_char -id=0 -type=eskimo -health=1 -field=0
  obtain
  dump_all

- **Expected output**
  1
  1
  0
  1
  c
  <>
  <0><0><1><3><{0, shovel}><eskimo>
  <0><true><-1><0><null><null><0>

8.2.14 Eskimo Obtains Food

- **Description**
  The Eskimo picks up a unit of food from the field he is standing on.
- **Checked functionality, expected failure points**
  The success rate of digging out a given tool from a given field:
  the Eskimo is actually standing in a field where:
  - number of snow layers 0
  - no building
  (- the desired object is indeed frozen in the field)
  If these are not met, the message "The device cannot be excavated!" is displayed.
  If the excavation is successful, the tool is added to the character's imaginary toolbox with the highest number.
- **Bento**
  create_game -blizzard=1 -rand=0
  create_field -id=0 -capacity=-1 -snow=0 -explored=1 -building=false -tool_category=s -tool_type=food -durability=-1
  create_char -id=0 -type=eskimo -health=1 -field=0
  obtain
  dump_all

- **Expected output**
  1
  1
  0
  1
  c
  <>
  <0><0><1><3><{0, food}><eskimo>
  <0><true><-1><0><null><null><0>

8.2.15 Eskimo Obtains Diving Gears

- **Description**
  The Eskimo puts on a wetsuit from the field he is standing on.
- **Checked functionality, expected failure points**
  The success rate of digging out a given tool from a given field:
  the Eskimo is actually standing in a field where:
  - number of snow layers 0
  - no building
  (- the desired object is indeed frozen in the field)
  If these are not met, the message "The device cannot be excavated!" is displayed.
  If the excavation is successful, the tool is added to the character's imaginary toolbox with the highest number.
- **Bento**
  create_game -blizzard=1 -rand=0
  create_field -id=0 -capacity=-1 -snow=0 -explored=1 -building=false -tool_category=s -tool_type=dgear -durability=-1
  create_char -id=0 -type=eskimo -health=1 -field=0
  obtain
  dump_all

- **Expected output**
  1
  1
  0
  1
  c
  <>
  <0><0><1><3><{0, dgear}><eskimo>
  <0><true><-1><0><null><null><0>

8.2.16 Eskimo Obtains Rope

- **Description**
  This test case is responsible for demonstrating and controlling the situation where an Eskimo operator picks up a rope.
- **Checked functionality, expected failure points**

A prerequisite for a successful move is that there is enough "space" in the actor's toolbox. If this is the case, then no other errors can occur during the run if the methods are correctly written.

- **Bemenet**
  create_game -blizzard=8 -rand=0
  create_field -id=0 -capacity=-1 -snow=0 -explored=1 -building=null -tool_category=s -tool_type=rope -durability=0
  create_char -id=0 -type=eskimo -health=4 -field=0
  active_char -id=0
  field_info -id=0
  obtain
  dump_all
- **Expected output**
  <0><true><-1><0><rope><null><0> 1
  1
  0
  8
  c
  <>
  <0><0><4><3><{0, rope}><eskimo>
  <0><true><-1><0><null><null><0>

8.2.17 Eskimo Interacts With Igloo

- **Description**
  This test case is responsible for presenting and controlling the situation where an Eskimo actor uses his skill and builds an igloo.
- **Checked functionality, expected failure points**
  For the move to be successful, there must not be another igloo in the field or snow on it. If these conditions are met, then no other errors can occur during the run if the methods are correctly written.
- **Bemenet**
  create_game -blizzard=8 -rand=0
  create_field -id=0 -capacity=-1 -snow=2 -explored=1 -building=null -tool_category=null
  create_char -id=0 -type=eskimo -health=4 -field=0
  active_char -id=0
  field_info -id=0
  ability
  dump_all
- **Expected output**
  <0><true><-1><0><null><null><2>
  1
  1
  0
  8
  c

<>
<0><0><4><3><><eskimo>
<0><true><-1><0><null><igloo><1>

8.2.18 Explorer Uses Ability

- **Description**
  This test case is responsible for representing and verifying the situation where an Explorer operator uses its capability and determines the capacity of a field.
- **Checked functionality, expected failure points**
  With correctly written methods, no errors should occur during execution.
- **Bento**
  create_game -blizzard=8 -rand=0
  create_field -id=0 -capacity=-1 -snow=0 -explored=0 -building=null -tool_category=null
  create_char -id=0 -type=explorer -health=4 -field=0
  active_char -id=0
  field_info -id=0
  ability
  dump_all
- **Expected output**
  <0><false><-2><0><null><null><0>
  1
  1
  0
  8
  c
  <>
  <0><0><4><3><><explorer>
  <0><true><-1><0><null><null><0>

8.2.19 Eskimo Uses Rope

- **Description**
  This test case is responsible for presenting and controlling the situation where an Eskimo actor uses his rope and goes out to someone.
- **Checked functionality, expected failure points**
  For the move to succeed, the Character to be saved must be on an adjacent field and actually drowning, and the Character to be saved must have a Rope. If these are met, then no other errors can occur during the run if the methods are correctly written.
- **Bemenet**
  create_game -blizzard=8 -rand=0
  create_field -id=0 -capacity=-1 -snow=0 -explored=1 -building=null -tool_category=null
  create_field -id=1 -capacity=0 -snow=0 -explored=1 -building=null -tool_category=null
  add_neighbour -id1=0 -id2=1
  create_char -id=0 -type=eskimo -health=4 -field=0
  create_char -id=1 -type=explorer -health=4 -field=1

drowning -id=1 -timeleft=2
addstdtool -location_id=0 -durability=10 -type=rope
dump_all
active_char -id=0
tool -id=0
1
dump_all

- **Expected output**

2
2
0
8
c
<1>
<0><0><4><4><{0,rope}><eskimo>
<1><1><4><4><explorer>
<0><true><-1><0><1><null><null><0>
<1><true><0><1><0><null><null><0>
Which character did you save?
1
1
0
8
c
<>
<0><0><4><3><{0,rope}><eskimo>
<1><0><4><4><explorer>
<0><true><-1><0, 1><1><null><null><0>
<1><true><0><0><null><null><0>

8.2.20 Polar Bear Kills Eskimo

- **Description**

  This test case is responsible for presenting and controlling the situation where an Eskimo character is hit and killed by a polar bear, resulting in the loss of the game.

- **Checked functionality, expected failure points**

  For the move to be successful, the Eskimo to be killed must not be standing in an igloo field. If this is the case, then no other errors can occur during the run if the methods are correctly written.

- **Bemenet**

create_game -blizzard=8 -rand=0
create_field -id=0 -capacity=-1 -snow=0 -explored=1 -building=null -tool_category=null
create_field -id=0 -capacity=-1 -snow=0 -explored=1 -building=null -tool_category=null
add_neighbour -id1=0 -id2=1
create_char -id=0 -type=eskimo -health=4 -field=0
create_char -id=0 -type=bear -health=1 -field=1

dump_all
skip
dump_all

- **Expected output**
  2
  1
  0
  8
  c
  <>
  <0><0><4><4><><eskimo>
  <0><true><-1><0><1><null><null><0>
  <1><true><-1><1><0><null><null><0>
  2
  1
  0
  7
  l
  <>
  <0><0><4><4><><eskimo>
  <0><true><-1><0, 1><1><null><null><0>
  <1><true><-1><><0><null><null><0>

8.2.21 Igloo Saves Eskimo From Polar Bear

- **Description**
  The polar bear steps onto the player's field, but because the player is iglooed, it does not die.
- **Checked functionality, expected failure points**
  Does the igloo really protect the player from the polar bear. It is a mistake if the bear kills the character on the igloo when it enters its field.
- **Bento**
  create_game -blizzard=10 -rand=0
  create_field -id=0 -capacity=-1 -snow=2 -explored=1 -building=igloo
  -tool_category=null -tool_type=shovel -durability=0
  create_field -id=1 -capacity=-1 -snow=2 -explored=1 -building=null
  -tool_category=null -tool_type=shovel -
  durability=0 create_char -id=0 -type=eskimo -
  health=4 -field=0 create_char -id=1 -type=bear -
  health=4 -field=1 skip
  dump_all

- **Expected output**
  2
  1
  0
  9
  c
  <>

<0><0><4><4><><eskimo>
<1><0><4><4><bear>
<0><true><-1><0, 1><1><null><igloo><2>
<1><true><-1><0><null><null><2>

8.2.22 Eskimo Uses Fragile Shovel

- **Description**
  The Eskimo shovels two layers of snow from the field with a single use of the fragile shovel.

- **Checked functionality, expected failure points**
  Is two layers of snow really disappearing from the field Does the number of energy points of the operator decrease. Does the lifetime of the shovel decrease.
  The test can only be carried out if there is a non-broken fragile shovel on the arm and at least two layers of snow in the field.

- **Bento**
  create_game -blizzard=10 -rand=0
  create_field -id=0 -capacity=-1 -snow=2 -explored=1 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_char -id=0 -type=eskimo -health=4 -field=0
  addstdtool -location_id=0 -durability=3 -type=fshovel
  field_info -id=0
  char_info -id=0
  tool -id=0
  field_info -id=0
  char_info -id=0

- **Expected output**
  <0><true><-1><0><null><null><2>
  <0><0><4><4><{0, fshovel,3}><eskimo>
  <0><true><-1><0><null><null><0>
  <0><0><4><3><{0, fshovel,2}><eskimo>

8.2.23 Eskimo Uses Fragile Shovel Too Many Times, Breaks It

- **Description**
  The Eskimo uses his fragile shovel three times, each time shoveling 2 layers of snow, and then for the fourth time he can only shovel one layer of snow because the shovel is broken.

- **Checked functionality, expected failure points** Indeed the right amounts of snow are disappearing from the field.
  Is the number of energy points of the operator decreasing.
  The test can only be carried out if there is a fragile shovel not used on the arm and the

field has 8 layers of snow.

- **Bemenet**
  create_game -blizzard=10 -rand=0
  create_field -id=0 -capacity=-1 -snow=8 -explored=1 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_char -id=0 -type=eskimo -health=4 -field=0
  addstdtool -location_id=0 -durability=3 -type=fshovel
  field_info -id=0
  char_info -id=0
  tool -id=0
  field_info -id=0
  char_info -id=0
  tool -id=0
  field_info -id=0
  char_info -id=0
  tool -id=0
  field_info -id=0
  char_info -id=0
  tool -id=0
  field_info -id=0
  char_info -id=0

- **Expected output**
  <0><true><-1><0><null><null><8>
  <0><0><4><4><{0, fshovel,3}><eskimo>
  <0><true><-1><0><null><null><6>
  <0><0><4><3><{0, fshovel,2}><eskimo>
  <0><true><-1><0><null><null><4>
  <0><0><4><2><{0, fshovel,1}><eskimo>
  <0><true><-1><0><null><null><2>
  <0><0><4><1><{0, fshovel,0}><eskimo>
  <0><true><-1><0><null><null><1>
  <0><0><4><0><{0, fshovel,0}><eskimo>

8.2.24 Eskimo Interacts With Tent

- **Description**
  The Eskimo builds a tent in his field.
- **Functionality checked, expected failure points**
  Whether the Eskimo has lost an energy point.
  Has the tent been built?
  The Eskimo must have a tent, there can be no buildings in the field to begin with.
- **Bemenet**
  create_game -blizzard=10 -rand=0
  create_field -id=0 -capacity=-1 -snow=0 -explored=1 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_char -id=0 -type=eskimo -health=4 -field=0
  addstdtool -location_id=0 -durability=0 -type=tent
  field_info -id=0

char_info -id=0
tool -id=0
field_info -id=0
char_info -id=0

- **Expected output**
  <0><true><-1><0><null><null><0>
  <0><0><4><4><{0, tent}><eskimo>
  <0><true><-1><0><null><tent><0>
  <0><0><4><3><><eskimo>

8.2.25 Tent Gets Broken After 1 Turn

- **Description**
  The Eskimo builds his tent. After completing his turn, the tent disappears.

- **Checked functionality, expected failure points**
  When the round is completed, does the tent really disappear?
  The Eskimo must have a tent, there can be no buildings in the field to begin with.

- **Bemenet**
  create_game -blizzard=10 -rand=0
  create_field -id=0 -capacity=-1 -snow=5 -explored=1 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_char -id=0 -type=eskimo -health=4 -field=0
  addstdtool -location_id=0 -durability=0 -type=tent
  tool -id=0
  field_info -id=0
  skip
  field_info -id=0

- **Expected output**
  <0><true><-1><0><null><tent><5>
  <0><true><-1><0><null><null><5>

Tests for the given track

1. Bear moves and eats someone, players just climb up and down

- **Bemenet**
  create_game -blizzard=30 -rand=0
  create_field -id=0 -capacity=2 -snow=0 -explored=0 -building=null -tool_category=k -tool_type=3 -durability=0
  create_field -id=1 -capacity=-1 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_field -id=2 -capacity=0 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_field -id=3 -capacity=-1 -snow=0 -explored=0 -building=null -tool_category=k -tool_type=1 -durability=0

create_field -id=4 -capacity=0 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=5 -capacity=-1 -snow=0 -explored=0 -building=null -tool_category=k -tool_type=2 -durability=0
create_field -id=6 -capacity=-1 -snow=0 -explored=0 -building=null -tool_category=s -tool_type=shovel -durability=0
create_field -id=7 -capacity=0 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=8 -capacity=1 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=9 -capacity=0 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=10 -capacity=2 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=11 -capacity=0 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=12 -capacity=-1 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -
durability=0 add_neighbour -id1=0 -id2=1
add_neighbour -id1=0 -id2=2
add_neighbour -id1=0 -id2=3
add_neighbour -id1=1 -id2=2
add_neighbour -id1=1 -id2=4
add_neighbour -id1=1 -id2=2
add_neighbour -id1=1 -id2=4id1=1 -id2=5
add_neighbour -id1=2 -id2=3
add_neighbour -id1=2 -id2=5
add_neighbour -id1=2 -id2=6
add_neighbour -id1=3 -id2=6
add_neighbour -id1=3 -id2=7
add_neighbour -id1=4 -id2=5
add_neighbour -id1=4 -id2=8
add_neighbour -id1=5 -id2=6
add_neighbour -id1=5 -id2=8
add_neighbour -id1=5 -id2=9
add_neighbour -id1=6 -id2=7
add_neighbour -id1=6 -id2=9
add_neighbour -id1=6 -id2=10
add_neighbour -id1=7 -id2=10
add_neighbour -id1=8 -id2=9
add_neighbour -id1=8 -id2=11
add_neighbour -id1=8 -id2=12
add_neighbour -id1=9 -id2=10
add_neighbour -id1=9 -id2=12
add_neighbour -id1=10 -id2=12
add_neighbour -id1=11 -id2=12
create_char -id=11 -type=bear -health=4 -field=1
create_char -id=81 -type=eskimo -health=4 -field=8
create_char -id=101 -type=explorer -health=4 -field=10

create_char -id=121 -type=explorer -health=4 -field=12

move -to=5
move -to=12
skip
move -to=6
skip
move -to=10
move -to=6
move -to=3
skip
move -to=6
dump_all

- **Expected output**

13
4
11
26
l
<>
<11><6><bear>
<81><12><4><4><eskimo>
<101><6><4><0><><explorer>
<121><3><4><4><explorer>
<0><false><2><1,2,3><r3><null><0>
<1><false><-1><>0,2,4,5><null><null><0>
<2><false><0><0,1,3,5,6><null><null><0>
<3><false><-1><121><0,2,6,7><r1><null><0>
<4><false><0><><1,2,5,8><null><null><0>
<5><false><-1><><1,2,4,6,8,9><r2><null><0>
<6><false><-1><11,101><2,3,5,7,9,10><a><null><0>
<7><false><0><3,6,10><null><null><0>
<8><false><1><4,5,9,11,12><null><null><0>
<9><false><0><><5,6,8,10,12><null><null><0>
<10><false><2><6,7,9,12><null><null><0>
<11><false><0><8,12><null><null><0>
<12><false><-1><81><8,9,10,11><null><null><0>

2. Players walk onto a fragile board and it breaks

- **Bemenet**

create_game -blizzard=30 -rand=0
create_field -id=0 -capacity=2 -snow=0 -explored=0 -building=null -tool_category=k -tool_type=3 -durability=0
create_field -id=1 -capacity=-1 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=2 -capacity=0 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=3 -capacity=-1 -snow=0 -explored=0 -building=null -tool_category=k -tool_type=1 -durability=0

create_field -id=4 -capacity=0 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=5 -capacity=-1 -snow=0 -explored=0 -building=null -tool_category=k -tool_type=2 -durability=0
create_field -id=6 -capacity=-1 -snow=0 -explored=0 -building=null -tool_category=s -tool_type=shovel -durability=0
create_field -id=7 -capacity=0 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=8 -capacity=1 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=9 -capacity=0 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=10 -capacity=2 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=11 -capacity=0 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=12 -capacity=-1 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -
durability=0 add_neighbour -id1=0 -id2=1
add_neighbour -id1=0 -id2=2
add_neighbour -id1=0 -id2=3
add_neighbour -id1=1 -id2=2
add_neighbour -id1=1 -id2=4
add_neighbour -id1=1 -id2=2
add_neighbour -id1=1 -id2=4id1=1 -id2=5
add_neighbour -id1=2 -id2=3
add_neighbour -id1=2 -id2=5
add_neighbour -id1=2 -id2=6
add_neighbour -id1=3 -id2=6
add_neighbour -id1=3 -id2=7
add_neighbour -id1=4 -id2=5
add_neighbour -id1=4 -id2=8
add_neighbour -id1=5 -id2=6
add_neighbour -id1=5 -id2=8
add_neighbour -id1=5 -id2=9
add_neighbour -id1=6 -id2=7
add_neighbour -id1=6 -id2=9
add_neighbour -id1=6 -id2=10
add_neighbour -id1=7 -id2=10
add_neighbour -id1=8 -id2=9
add_neighbour -id1=8 -id2=11
add_neighbour -id1=8 -id2=12
add_neighbour -id1=9 -id2=10
add_neighbour -id1=9 -id2=12
add_neighbour -id1=10 -id2=12
add_neighbour -id1=11 -id2=12
create_char -id=121 -type=explorer -health=4 -field=12
create_char -id=11 -type=bear -health=4 -field=1
create_char -id=81 -type=eskimo -health=4 -field=8

```
create_char -id=101 -type=explorer -health=4 -field=10
```

move -to=8
dump_all

- **Expected output**
  13
  4
  121
  30
  c
  <81,121>
  <11><1><4><4><explorer>
  <81><8><bear>
  <101><10><4><4><eskimo>
  <121><8><4><4><explorer>
  <0><false><2><1,2,3><r3><null><0>
  <1><false><-1><11>0,2,4,5><null><null><0>
  <2><false><0><0,1,3,5,6><null><null><0>
  <3><false><-1><0,2,6,7><r1><null><0>
  <4><false><0><><1,2,5,8><null><null><0>
  <5><false><-1><><1,2,4,6,8,9><r2><null><0>
  <6><false><-1><><2,3,5,7,9,10><a><null><0>
  <7><false><0><3,6,10><null><null><0>
  <8><false><1><81,121><4,5,9,11,12><null><null><0>
  <9><false><0><><5,6,8,10,12><null><null><0>
  <10><false><2><101><6,7,9,12><null><null><0>
  <11><false><0><8,12><null><null><0>
  <12><false><-1><8,9,10,11><null><null><0>

3. Eskimo builds igloos

- **Bento**
  create_game -blizzard=30 -rand=0
  create_field -id=0 -capacity=2 -snow=0 -explored=0 -building=null -tool_category=k -tool_type=3 -durability=0
  create_field -id=1 -capacity=-1 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_field -id=2 -capacity=0 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_field -id=3 -capacity=-1 -snow=0 -explored=0 -building=null -tool_category=k -tool_type=1 -durability=0
  create_field -id=4 -capacity=0 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
  create_field -id=5 -capacity=-1 -snow=0 -explored=0 -building=null -tool_category=k -tool_type=2 -durability=0
  create_field -id=6 -capacity=-1 -snow=0 -explored=0 -building=null -tool_category=s -tool_type=shovel -durability=0
  create_field -id=7 -capacity=0 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0

create_field -id=8 -capacity=1 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=9 -capacity=0 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=10 -capacity=2 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=11 -capacity=0 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -durability=0
create_field -id=12 -capacity=-1 -snow=0 -explored=0 -building=null -tool_category=null -tool_type=shovel -
durability=0 add_neighbour -id1=0 -id2=1
add_neighbour -id1=0 -id2=2
add_neighbour -id1=0 -id2=3
add_neighbour -id1=1 -id2=2
add_neighbour -id1=1 -id2=4
add_neighbour -id1=1 -id2=2
add_neighbour -id1=1 -id2=4id1=1 -id2=5
add_neighbour -id1=2 -id2=3
add_neighbour -id1=2 -id2=5
add_neighbour -id1=2 -id2=6
add_neighbour -id1=3 -id2=6
add_neighbour -id1=3 -id2=7
add_neighbour -id1=4 -id2=5
add_neighbour -id1=4 -id2=8
add_neighbour -id1=5 -id2=6
add_neighbour -id1=5 -id2=8
add_neighbour -id1=5 -id2=9
add_neighbour -id1=6 -id2=7
add_neighbour -id1=6 -id2=9
add_neighbour -id1=6 -id2=10
add_neighbour -id1=7 -id2=10
add_neighbour -id1=8 -id2=9
add_neighbour -id1=8 -id2=11
add_neighbour -id1=8 -id2=12
add_neighbour -id1=9 -id2=10
add_neighbour -id1=9 -id2=12
add_neighbour -id1=10 -id2=12
add_neighbour -id1=11 -id2=12
create_char -id=81 -type=eskimo -health=4 -field=8
create_char -id=11 -type=bear -health=4 -field=1
create_char -id=101 -type=explorer -health=4 -field=10
create_char -id=121 -type=explorer -health=4 -field=12
ability -field=8
dump_all

- **Expected output**
  13
  4
  121
  30

c

<>
<11><1><4><4><eskimo>
<81><8><4><3><bear>
<101><10><4><4><explorer>
<121><12><4><4><explorer>
<0><false><2><1,2,3><r3><null><0>
<1><false><-1><11>0,2,4,5><null><null><0>
<2><false><0><0,1,3,5,6><null><null><0>
<3><false><-1><0,2,6,7><r1><null><0>
<4><false><0><><1,2,5,8><null><null><0>
<5><false><-1><><1,2,4,6,8,9><r2><null><0>
<6><false><-1><><2,3,5,7,9,10><a><null><0>
<7><false><0><3,6,10><null><null><0>
<8><false><1><81><4,5,9,11,12><null><igloo><0>
<9><false><0><><5,6,8,10,12><null><null><0>
<10><false><2><101><6,7,9,12><null><null><0>
<11><false><0><8,12><null><null><0>
<12><false><-1><121><8,9,10,11><null><null><0>

8.3 Plans for programmes to support testing

During the run, the SchParser class will run and support the testing. The class contains a method that can compare the output of a run with the corresponding expected output (if such a predefined file exists). Testing can be done through console manually, or with a predefined input file, which can be selected when the 'conf' console command is given (similar to the way the output file is specified). After the 'test' command, only the sample test case with the specified sequence number is run, otherwise the program runs until the game is over.
Depending on the parameter following the 'test' command, test_XY_in.txt is always run, the output of which is compared with the test_XY_out.txt file. Following the 'test' command, if it finds a deviation from the expected, it will specify exactly which 'attribute(s)' it found a problem with and what it should have been.
Accordingly, there are three types of termination messages after the program has run:

1.) The test has been run, the game is over, but no comparison has been made, so nothing is written
2.) A comparison has been performed and it was successful, so we get the message "Successful testing"
3.) A comparison has been made and an error has been found, in which case the message "Failed testing" is displayed, as well as the name of the attribute(s) with the error and their expected value

8.4 Diary

Start at	Duration	Participants	Description
2020.04.07. 14:15	1,5 hours	Bankó Bankó Hain Juhász Lukács	Conference.
2020.04.10. 21:00	1 hour	Bankó Bankó Hain Juhász Lukács	Conference.
2020.04.12. 20:00	1,5 hours	Bankó Bankó Hain Juhász Lukács	Conference.
2020.04.10. 15:00	3 hours	Banker	Preparing class descriptions for Tool and his descendants.
2020.04.12. 14:00	3 hours	Banker	6-11 Preparing test cases
2020. 04. 10	1 hour	Retrieved from	Make changes to the output language
2020. 04. 11	3 hours	Retrieved from	Redesign of character and descendants and preparation of class descriptions
2020. 04. 12	2 hours	Retrieved from	11-16 Preparing test cases + checking
2020.04.12	2 hours	Digging	Redesigning class structure and methods
2020.04.12	3 hours	Digging	1-5 + 3 pre-defined test cases
2020.04.13	2 hours	Digging	Improving test cases, creating class descriptions, state charts
2020.04.12	3 hours	Lukács	Game class description update,

			writing pseudocodes, test writing
2020.04.13	2 hours	Lukács	Test repair, verification